Analyzing NFS Client Performance with IOzone

Don Capps (don.capps@hp.com)
Tom McNeal (trmcneal@comcast.net)

IOzone is a benchmarking tool useful for analyzing file system performance on a number of different platforms, including Linux, HP-UX, Solaris, and many others. It uses file system IO as its primary load generation, presenting the systems under test with a large range of file IO requests, running from small to very large file sizes, with varying record sizes, while performing a number of different file access patterns. It also permits the benchmark administrator to modify system test parameters, affecting the underlying system response to these differing access sizes and patterns in order to illustrate system issues that might not otherwise be obvious, or which might vary from platform to platform.

In addition to measuring local file system behavior, IOzone may be used for analyzing client performance in an NFS client/server environment, when using NFS mounted file systems as the target for its IO request set. This paper will discuss the usage of IOzone as a performance analysis tool for NFS clients. It will cover the complete IO request set, possible options for varying system response, and the possible arrangement of output data that will be most illustrative of the system behavior. In addition, it will discuss general system parameters as well, in order to learn the most about system response to the demands of the benchmark.

Finally, note that IOzone, plus accompanying documentation, may be downloaded from www.iozone.org.

File System IO Requests

The benchmark will execute file IO to a broad range of file sizes, from 64Kbytes to 512Mbytes in the default test case, using record sizes ranging from 4Kbytes to 16Mbytes. For both files and records, each size increase will simply be a doubling of the previous value. If requested, the benchmark will also increase the file sizes to an optional larger value, in order to allow testing of large file systems supported by 64-bit platforms. Note that the word "test", the term used in this paper and in the IOzone documentation, refers to a single type of access - a sequential write, from the beginning to the end of the file, for example - executed against all files of the given file sizes defined by the command line. When executing the IO on each file, it will use the following access patterns:

- Sequential read/write – The records will be written to a new file, and read from an existing file, from beginning to end.
- Sequential rewrite – The records will be rewritten to an existing file, from beginning to end.
- Sequential reread – The records of a recently read file will be reread, from beginning to end. This usually results in better performance than the read, due to cached data.
- Backwards read – The records will be read from the end of the file to the beginning. Few operating systems optimize file access according to this pattern, and some applications, such as MSC Nastrans, do this often.
- Random read/write – The records, at random offsets will be written and read.
- Record rewrite – A record will be rewritten in an existing file, starting at the beginning of the file.
- Strided read – Records will be read at specific constant intervals, from beginning to end.
- fread/fwrite – The fread() and fwrite() functions will be invoked to write to a new file, and read from an existing file, from beginning to end.
- freread – The fread() function is invoked on a recently read file.
- frewrite – The fwrite() function is invoked on an existing file.
- pread/pwrite – The performance of pread() and pwrite() on systems that support these system calls will be measured.

Note that when running the benchmark under normal circumstances, it is important to limit the size of the buffer cache to something less than the maximum file size, in order to see the effect of cache misses on the benchmark. This is, of course, not relevant to memory mapped files, but note that when reviewing the benchmark results, you may see the performance effects of other caches, in addition to the buffer cache, when executing IO requests within those cache limits. These will include the processor cache and secondary memory caches, as well as the buffer cache. Assuring that the larger file sizes exceed these limits will accurately illustrate the effects of cache hits and misses.

Standard setups for NFS Clients

A common configuration for NFS client analysis with IOzone includes the following features:

- IOzone is executed from a locally mounted directory
- File IO is limited to files contained in NFS mounted directories
- Client caches are cleared out by unmounting and remounting the file system between tests ("***-U mountpoint***")
- Commit time for NFS V3 is included in the measurements by including file closure times ("***-c***")
- File sizes run from 64Kbytes to 512Mbytes, using record sizes from 4Kbytes up to 16Mbytes ("***-a***")
- When the file size exceeds 32Mbytes, the record size will have a lower limit of 64K. This limit may be removed with the "***-z***" option
- Aside from the above limit, all record sizes are used for those files which can contain them (e.g., 64K files will be tested with record sizes running from 4K to 64K, doubling in size with each iteration)
- Reports are generated in Excel spreadsheet format ("***-R***")
- The Excel spreadsheet format is used to generate graphical images in a designated file *(*"***-b filename***").
- Results from standard out are piped into a local file.

The IOzone command which accomplishes the above, assuming */mnt/foo* is an NFS mounted directory, and that the local directory is locally mounted, is as follows:

```
./iozone -azcR -U /mnt/foo -f /mnt/foo/testfile -b exceloutput.xls > logfile
```

System Level Variations

Several variations which account for system level features of the client may be invoked through benchmark parameters. These should be used as variants for data gathering in addition to the standard runs, in order to analyze system level effects of client capabilities, particularly when analyzing SMP clients. These variations include:

- Processor affinity of all threads/processes ("***-P #*** ")
- Limit on the least number of processes ("***-l n***")
- Processor cache purges ("***-p***")
- CPU cache size and cache line size ("***-S size***" and "***-L size***")
- Removal of "stonewalling", an internal feature of IOzone ("***-x***")

All of these should be carefully considered before adding them to the benchmark configuration. The one which may have the most benefit from the system point of view is the invocation of processor affinity, guaranteeing that processes will be distributed evenly among a set number of processors on the SMP client, starting with processor # n, as given in the parameter list. You may also invoke the assurance that all processor caches are purged at the end of each test, and that a certain number of processes will be running as part of the benchmark.

In addition to purging caches, you may wish to manage system cache behavior to some extent, as mentioned in the previous section. Depending on various system characteristics, you should limit the size of the buffer cache, and also the size of memory and processor caches, in order to obtain a clear picture of system behavior that is not masked by cache effects.

Another variation that involves the internal design of IOzone, is the request to remove an internal feature called "stonewalling", which refers to the attempt to launch all threads/processes involved in a single test at the same time. Stonewalling also halts all threads/processes when one has completed, and recalculates the IO times based on all currently completed IO. This assures that the system is exposed to a consistent process load at all phases of the tests, and that a single IO request encountering system delays unique to that request does not distort the overall results of the benchmark. Since the causes of such delays may need to be investigated, regardless of the effect on the overall benchmark, the option to remove this constraint is available.

File System Behavior Variations

In addition to system level variations, the behavior of the file system is critical to the results seen in the benchmark; perhaps much more critical than the system level variants. The most commonly encountered variant calls for complete synchronous IO, using the "O_SYNC" option when opening all files under test. This variant requires that all IO requests should be sent immediately to stable storage, which, in the NFS client case, is the disk on the server. This should, of course, force NFS V2 and NVS V3 to behave identically; in this case, even the usual V3 commit at file close time should not take place, since the client will have already demanded that it receive the "NFS_FILE_SYNC" reply status from the server for all IO requests, forcing the server to send each individual request to stable storage, and making the final commit request unnecessary.

Another commonly used feature is file locking; the benchmark will allow the administrator to force locking for all file accesses. This has a side effect of causing most NFS implementations to disable the client and server caches. As before, this should be used when comparing runs to the standard run given above.

The benchmark administrator may choose to use memory mapped file IO, if the client supports memory mapping when opening files exported from an NFS server. Since memory mapped IO may not result in the occurrence of any immediate file IO, the administrator may also choose to control the client's file access behavior by specifying synchronous IO with the IO_SYNC option, or by invoking the *msync()* system call, with the MS_SYNC option, which will guarantee a file IO call, along with the subsequent wait for IO completion. If desired, asynchronous IO might also be requested with the MS_ASYNC option, allowing internal system mechanisms to control data transfers to the server. Note that this does not emulate asynchronous exports of an NFS server, available on some platforms. These features are client based, and control IO access requests from the client to the server.

Another control of client IO, which may be considered equally complex and possibly dangerous in an NFS client, is POSIX asynchronous IO, implemented by calls to POSIX routines such as **aio_read(), aio_write(),** and **aio_error()**.

Finally, the Benchmark administrator may choose to include the delays incurred by *fsync()* and *fflush()* calls. This will reduce the NFS V3 client side effects due to caches. This is particularly useful when comparing different platforms, if one wishes to eliminate cache effects and concentrate on other platform differences. To reiterate, the interesting file system variants are

- Total synchronous file access, using the O_SYNC option when creating/opening files ("***-o***")
- Files locking used for all IO requests ("***-W***")
- Memory mapped file access (either synchronous or asynchronous mode) used in the client ("***-B***")
- POSIX Asynchronous IO used in the client, which should not be confused with Asynchronous IO implemented on some Servers, such as Linux and HP-UX.)) ("***-H n***" or "***-k n***")
- Include fsync and fflush calls in the timings ("***-e***")

Network Variations

When setting up clients and servers for benchmark analysis, many network configuration variations may be selected in order to illustrate system behavior across a range of conditions. The most significant ones with regards to NFS client performance analysis are:

- Network protocol – TCP or UDP
- Network speed, duplex settings, and auto-negotiation
- Client transfer size settings (***rsize*** and ***wsize***)
- Gigabit Ethernet and jumbo frames
- Number of stream heads
- Number of networking buffers for sockets

The most obvious may be which network protocol to select – TCP or UDP – when mounting the file system on the client. As is well known, TCP will impose a slight performance cost in an ideal, loss-free network, but given that such clean networks are rare, and may not even exist in a non-isolated network environment, the advantages of TCP's handling of the network transactions at the lower levels of the network stack, rather than up at the RPC layer with UDP, creates a significant advantage, particularly when confronted with lossy networks and the resulting cost of the inevitable retransmissions occurring in such networks. It may be useful to determine the actual cost of using TCP in an isolated environment, in comparison with UDP, but that should be a decision made by the individual benchmark analyst.

One issue that should be studied carefully is assuring that the network components negotiate full duplex behavior, either by explicit settings, or by auto-negotiation. An environment which accidentally defaults to half-duplex will introduce network collisions which are somewhat harmful to TCP performance, but deadly to UDP performance. The management of collisions are the single worst performance inhibitor for NFS, and must be avoided whenever possible, whether by using the correct duplex settings, or by eliminating network bottlenecks in intervening switches or servers, or by addressing any other network performance issue.

Given optimal settings for the network environment, the specific values of the NFS client's transfer sizes – the sizes set in the ***rsize*** and ***wsize*** parameters of the mount command – should be studied by the benchmark administrator, and may require that the benchmark be run at a variety of settings.

When using Gigabit Ethernet, many clients, servers, and switches now support jumbo frames. This will be particularly helpful when using the largest transfer sizes allowed, such as 32K, or even 64K, limiting the network packet fragmentation necessary for accomplishing each request. Note that the standard MTU size is 1500 bytes, and the Jumbo Frame MTU size is 9000 bytes.

New Features

Several new parameters have been added to IOzone, and may be invoked with a special parameter syntax, using the standard minus sign, and then a plus sign before giving the particular parameter letter. Please be sure to use both the minus and the plus sign, since the new parameters will be confused with old parameters if this difference is not seen.

One new feature is special debugging mode, invoked for error detection if the benchmark administrator suspects either internal benchmark problems, or platform data corruption problems. IOzone normally does a minimal amount of data validation so that the intrusion in the measurement is minimized. If one suspects a data integrity problem then one can enable the "***-+d***" option. This is activates a very intensive data validation model inside of IOzone.

Another feature of IOzone is the ability to test the aggregate throughput of an NFS server as seen by multiple NFS clients. This is called “distributed mode” or “cluster mode” and is activated using the “***-+m filename***” option).

The file "filename" contains lines in the following format:

client_name	***work_dir_on_client***	***path_to_IOzone_on_client***
client1.home.org	/home/foo	/home/foo/iozone
client2.home.org	/home/foo	/home/foo/iozone

Now, if you were to do something like:

iozone -t 2 -r 64 -s 1024 -+m filename

IOzone would then conduct throughput testing across all of the clients in the file "filename" Each client will start IOzone in the directory specified in the "filename" and run IOzone with 64kbyte records for a file of size 1 Mbyte. The collective throughput result will be displayed.

How does this magic work ?

IOzone, when invoked with the “***-+m***” option, becomes the controlling process which coordinates distributed IOzone testing, and does not itself do any direct I/O. It uses remsh/rsh to start IOzone on each of the clients. Each client's IOzone notifies the controlling process of its startup and awaits instructions on what to do. The controlling process tells each of the clients which test to run. The clients begin the selected test, run to a barrier and inform the controlling process that it is at the barrier. When the controlling process sees that all of the clients are at the barrier then it tells them to start. When the clients finish the test they send their results back to the controlling process. When any client finishes it’s work it will send notification to the controlling process. If stonewalling is enabled (Enabled by default. See –x flag) then the controlling process will contact all of the clients and request their current results. This mechanism is used to guarantee that the measurement was taken only during a period of time when all of the clients were running in parallel. When all of the clients are finished the controlling process then calculates the throughput and provides the results.

The software model in use is similar to a distributed shared memory model. The controlling process has the master copy and uses sockets to collect and distribute data to the clients so that they can update their copy. The clients send their changes back to the controlling process so that it has the most recent changes and is coherent. The underlying communication model provides for barriers, synchronous data flow, and asynchronous data flow.

Things you need to make it work:

- IOzone must be able to launch a rsh/remsh onto the clients in the client file "filename" without being challenged for a password. (rhosts would be a good thing here)
- The clients must be able to contact the controlling host, where you started IOzone, by its hostname. (assuming DNS is working)
- IOzone must be able to contact the clients by their names that are in the client file. (assuming DNS is working)
- Revision 3.74 or later of IOzone.

Finally, note that you should avoid running out of space on the clients. Error handling is not robust in this mode.

Interesting system parameters

A number of system parameters will have an effect on the system performance results measured by IOzone. The following is a short list of several system values which have proved to be interesting. Any additions to this list would be welcome information.

Number of biod daemons –

The number of biod daemons can have a significant impact on the throughput of an NFS client. These daemons are used to perform I/O from the client application to the NFS server. In some cases these daemons provide read-ahead and write behind for the client application. .

Number of nfsd daemons –

The number of nfsd daemons can have a significant impact on the throughput that the NFS server is providing to the client. The number of server daemons must be carefully monitored in order to make client comparisons valid.

Number of inodes/vnodes –

Both the server and client operating systems keeps a cache of inode/vnodes and/or file handles, as well as rnodes, in the case of the client. These cached structures can satisfy file attribute requests quickly, but if the number of these is too small then the operating system will have to perform real disk activity to re-read the attributes of the files.

Size of the name lookup cache –

If the server's name lookup cache is too small then real physical I/O operations may be required when they could have been avoided. This is a tunable that is more likely to need to be tuned if the NFS server has a large number of files that are being served to the clients.

Size of the network buffers for udp and tcp –

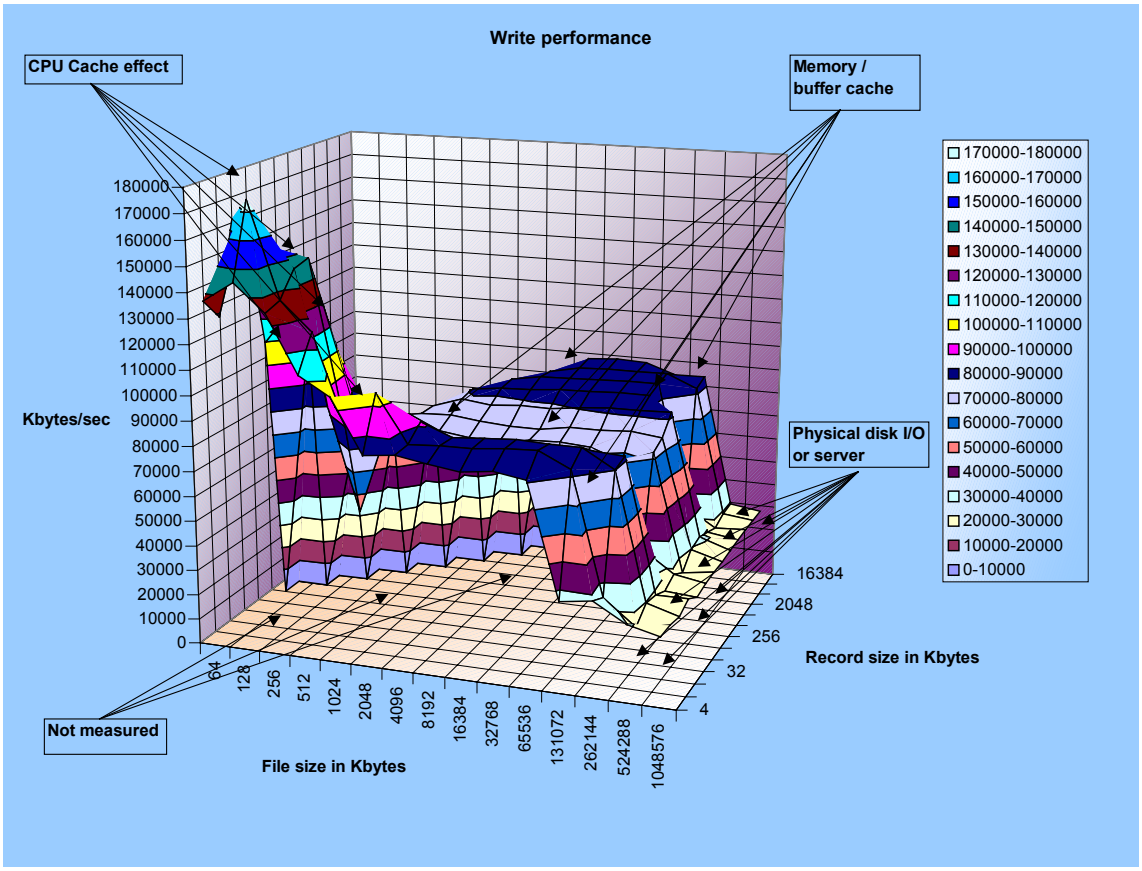
In reviewing server performance tuning activities logged in the SPECsfs benchmark submittals, the defined sizes of the server's network buffers was increased for many benchmark runs. This would have an effect on the client network, as well, so it is recommended that this value be reviewed. In linux, the value names – which can be tuned by writing to the "proc" file system, a subdirectory in the local file system which manages system parameters - are known as *"wmem_max", " rmem_max"," tcp_wmem" and " tcp_rmem".*

Displaying Results

The default results file will give the test results in tabular text format. For graphical representation, the benchmark administrator should request that the benchmark should also generate results in an Excel spreadsheet format ("***-R***" for output directed to standard out, and "***-b filename***" for output in the designated file . With the Excel spreadsheet, you will see the tabular results as seen in the default format, plus the Excel output for the individual tests, labeled as "iozone.runlog". In addition, the surface graphs generated by the Excel Chart Wizard, using the standard "Surface" plot set to the 3-D surface representation, may be seen under the names of each individual test – that is, "Write", "Rewrite", "Read", Reread", etc. This gives a very clear picture of the behavior of the system for the individual tests.

The IOzone web site contains an example spreadsheets, including some analyses of 8K vs. 32K transfer sizes for both UDP and TCP, which you may review at http://www.iozone.org/src/current/ . An example, showing features such as the effects of CPU cache size, memory cache size, and disk IO delays, is given below by a test run on a local file system. The examples at the given web site also have these effects labeled, and should provide an interesting illustration of the graphical capabilities of IOzone.

Example of local filesystem results



-

Summary

The IOzone benchmark is a useful and easily used tool for measuring file system performance, both in the local file system context, and for NFS clients. The definition of a standard run is easily determined and executed, and the tool provides many variants for focusing on specific features of both the NFS client, and of the NFS Server as well, since the server will affect the overall client behavior in several specified ways.

In addition to the specified command line instructions given in this paper, IOzone may also be run, on the Linux operating system, from a GUI-based interface embedded in an NFS test and measurement tool currently published on the Sourceforge software foundary, www.sourceforge.net. The tool may be downloaded from its Sourceforge home page, http://sourceforge.net/projects/nfstestmatrix/.

Finally, when running IOzone, it is important to gather baseline data first, prior to looking at system variations. With the baseline data as a fixed beacon, you should be able to study the behavior of your system, modify its behavior with any available methods, and determine the real effect of such modifications.